SCALING OVERLAPPING COMMUNITY DETECTION ALGORITHMS

BY

AMEY SHANKARRAO CHAUGULE

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2014

Urbana, Illinois

Adviser:

Professor Constantine Polychronopoulos

Abstract

Community structure is observed in many real-world networks in fields ranging from social networking to biological networks. Over the last decade many approaches have been proposed to efficiently detect the underlying structure of communities in graphs with a greater degree of correctness. This specific area of graph mining is known as community detection. It is one of the most critical components of graph mining. It helps in understanding the underlying properties of the graph.

While a lot of effort has been expended on detecting standard partitions in a graph, real world applications are complicated and they exhibit nodes belonging to multiple communities concurrently. Although many algorithms such as *Clique Percolation Method*, *COPRA,* etc., have been developed to detect overlapping communities in graphs, they rely on expensive and complicated optimization techniques and do not scale well to real world datasets containing millions of nodes and tens of millions of edges.

In this thesis, we propose a vertex centric approach to the problem and we evaluate scalability of overlapping community detection algorithms when implemented using vertex centric graph processing frameworks such as GraphLab/GraphChi. In particular we implemented BigCLAM in GraphChi and were able to achieve speedups of up to 6.5x on large scale real world graphs, thus proving that our approach can give linear speedups on the same hardware settings.

Acknowledgments

I owe the completion of this project to the love and support of many people. First, I thank my adviser, Prof. Constantine Polychronopoulos, who believed in me and championed my case to the graduate admissions committee. I also thank Dr. Konstantinos Karantasis, without whose mentorship this project would not have been conceived. I also thank my family and numerous friends such as Justin Martin, David S. Olsen and family, Sue Zhang, Richard Lauber, Shea Ferguson, Anthony Wojkowski, Dr. Ellick Chan, Kate Trader and her family, Steve Seo, Valentin Sidea, Jay Puntham-Baker, David Kaplinsky, Julie Cai, Shivaram Venkataraman, Adam Berry, Mulyanto and Jessica Poort, Mirko Montanari and many more whose love and friendship sustained me throughout the six years I spent at Illinois.

Contents

1. Introduction 1
2. Literature Review 6
2.1 Non-overlapping Community Detection 6
2.2 Overlapping Community Detection 8
3. The BigCLAM algorithm 13
3.1 Introduction 13
3.2 Community Detection using BigCLAM 14
3.2.1 Initialization 15
3.2.2 Determination of Community Affiliation 15
3.3 Algorithm 16
3.4 Performance 17
3.5 Qualitative Analysis 17
3.6 Possible Improvements 18
4. BigCLAM on GraphChi 20
4.1 Introduction 20
4.2 BigCLAM Implementation 21
5. Experimental Evaluation 23
5.1 Datasets and Hardware 23
5.2 Experiments on BigCLAM OpenMP 24
5.3 Experiments on BigCLAM GraphChi 26
6. Conclusion and Future Work 28
References 29

1. Introduction

Graphs are a natural and a very succinct way to represent a wide variety of systems found in nature. Social, biological and information networks lend themselves to be easily described as graphs and can be easily studied using graph theory constructs. Recent researches by mathematicians have focused on using statistical properties such as small world effect, degree distributions and communities/clusters, which these networks share with each other in order to study them [1].

Although there no clearly agreed upon definition of what a community is, most researches tend to consider a community within a network to be a subset of nodes with dense links among themselves while the number of links with other nodes in the network is quite sparse. Even so, the definition of what constitutes a community for a given graph tends to depend heavily on the particular problem domain represented by the graph.

Community detection has many direct real world applications. Clustering users in communities of similar interest is an intrinsic part of recommendation systems ubiquitous throughout the consumer web application space. In parallel computing, scheduling algorithms need to partition tasks among different processors in order to minimize communication between them. Various community detection/graph clustering approaches have been successfully used to solve this problem [2]. Outside the world of computer science, community detection has been used to identify major link nodes in

real world people networks in order to study the flow of information within organizations. The wealth of email communications obtained in the fallout of the Enron scandal has been used by researchers to study organizational and communication networks [3]. In biology, community detection is used to study networks that map interactions between proteins and in metabolic networks [4].

Many different algorithms have been proposed to detect communities in graphs for a wide variety of domains as catalogued by Fortunato et al. in their seminal survey in the field [4]. However, most community detection algorithms concern themselves with finding disjoint communities in a graph, i.e. a particular node can be a member of at most one partition. Many real world networks tend to contain overlapping communities where a particular node may be a part of multiple communities. In social networks for instance, one is a part of multiple communities or social circles like family, school friends, colleagues etc.

The term “overlapping community detection algorithms” refers to the class of community detection algorithms that allow nodes to belong to multiple communities at the same time. Kelley et al. showed that overlapping communities are a major feature of many real world social and biological networks [5]. Hence finding overlapping communities in graphs has more direct real world implications than mere disjoint community detection. While the problem of basic community detection has been looked at from many different perspectives and is generally well understood for a particular domain, the art of detecting overlapping communities in graphs remains a

nascent field of study. Several popular methods such as Link Clustering [6] and Clique Percolation Method [7] have been proposed in the area in recent years. However, as Yang and Leskovec show in [8], these methods are unable to scale well for graphs with more than hundreds of thousands of edges when benchmarked against real world datasets containing ground truth community structure. Given the time complexity of their computational kernels, which involve complex optimizations, these algorithms are unsuitable for very large real world graphs.

In the same paper, Yang and Leskovec propose the BigCLAM method and demonstrate state-of-the-art results when compared with Link Clustering and Clique Percolation [8]. The algorithm maximizes an objective function using non-negative matrix factorization to compute the affiliation matrix mapping nodes to communities. When the algorithm converges, the affiliation matrix obtained contains a latent factor for each node-community pair, showing the degree of belonging of the node for that community. BigCLAM is a considerable improvement over previous community detection algorithms and was able to achieve state-of-the-art performance on most benchmarks. Its performance only starts degrading for graphs with 10^9 edges. While BigCLAM is certainly an exciting development, parallelism and exploiting current generation frameworks for parallel and distributed computing is barely touched upon in the community detection algorithms [9]. This remains a critical area for work on this topic.

In order to efficiently process computations on large graphs, systems like Pregel from Google and GraphLab/GraphChi have been developed in recent years [10], [11]. In their

vertex centric processing model, programs are represented as a series of iterations in each of which the vertices of your graph can receive messages from the previous iteration, send messages to its neighbors and modify its own state and that of its outgoing edges or mutate the topology of the graph [10].

In this thesis, we recast the BigCLAM algorithm into this model using GraphChi and compared its scalability with an OpenMP implementation of the algorithm. We observed impressive speedups with this approach when compared to the baseline. In addition we demonstrated that our approach is vastly superior given that it is relatively trivial to port and distribute GraphChi programs onto GraphLab clusters, which can then be scaled to very large datasets with ease, thus proving the efficacy of using graph-processing systems for scaling overlapping community detection. Thus GraphChi enables us to have transparent linear speedups.

Given that graph theory terminology is often confusing and terms are used interchangeably, in the rest of this thesis the following terms are interchanged and have the same meaning:

1. Graphs and networks.
2. Communities, clusters and partitions.
3. Nodes and vertices.
4. Community detection and graph clustering.

The rest of this thesis is organized as follows:

Chapter 2 covers the literature review, Chapter 3 covers BigCLAM and its algorithmic implementation, Chapter 4 describes the GraphChi version while Chapter 5 contains the evaluation of two implementations, and we conclude our observations in Chapter 6.

2. Literature Review

In this section we will briefly give a broad overview of community detection algorithms and especially the current work in community detection algorithms.

2.1 Non-overlapping Community Detection

The primary element of the problem itself is ill-defined and there is a great degree of arbitrariness on even what constitutes a community. Naturally, the literature in the field is replete with a vast number of methods to tackle the problem. Some of the techniques that have been employed previously have been: modularity maximization, hierarchical clustering and random walks [4].

Modularity maximization forms the basis of the largest class of community detection algorithms. Modularity measures the strength of a given partition within a graph, it is the fraction of edges in in a partition that fall within the partition with subtraction of the expected density of edges according to the given graph model such as the Erdős–Rényi model [12], [13]. It thus reflects the concentration of edges within a community compared to the expected distribution of edges between all nodes of the graph. Optimizing your community detection algorithm to maximize modularity within the graph thus guarantees optimal partitions. Maximizing modularity is an NP-complete problem as work by Brandes et al. have shown [14] since there are innumerable ways in

which the graph could be partitioned. The polynomial solution to the problem becomes almost impossible as the size of the graph increases [4].

Newman was first to use the modularity maximization approach for community detection [15]. He devised a greedy algorithm based on agglomerative hierarchical clustering, wherein smaller communities of vertices merge to form larger communities such that modularity is maximized [4]. The algorithm converges when the calculations for subsequent merge steps do not cause an increase in modularity. This algorithm calculates modularity on the entire graph leading to many useless operations. Many other algorithms in this class concentrate on improving the speed and efficiency of the core modularity computation. Some employ techniques like *simulated annealing*, which measure fluctuations in the graph to calculate modularity [16], while others use external optimization techniques [17] or spectral clustering [18].

A *Random Walk* is a formalized mathematical representation of a randomized graph traversal [19]. It is heavily used in applied graph theory, most famously to estimate the size of the web. The basic idea here is that if a graph contains a strong community structure, the random walker spends more time inside a community instead given the higher density of internal edges than external [4]. Zhou first demonstrated applicability of random walks to the problem of community detection [20]. He states that the distance between two nodes, say Node A and Node B, in a graph to be the average number of edges a random walker needed to get from Node A to Node B. The closer the

nodes are to each other by this distance measure, the more likely they are to belong to the same community [4], [20].

2.2 Overlapping Community Detection

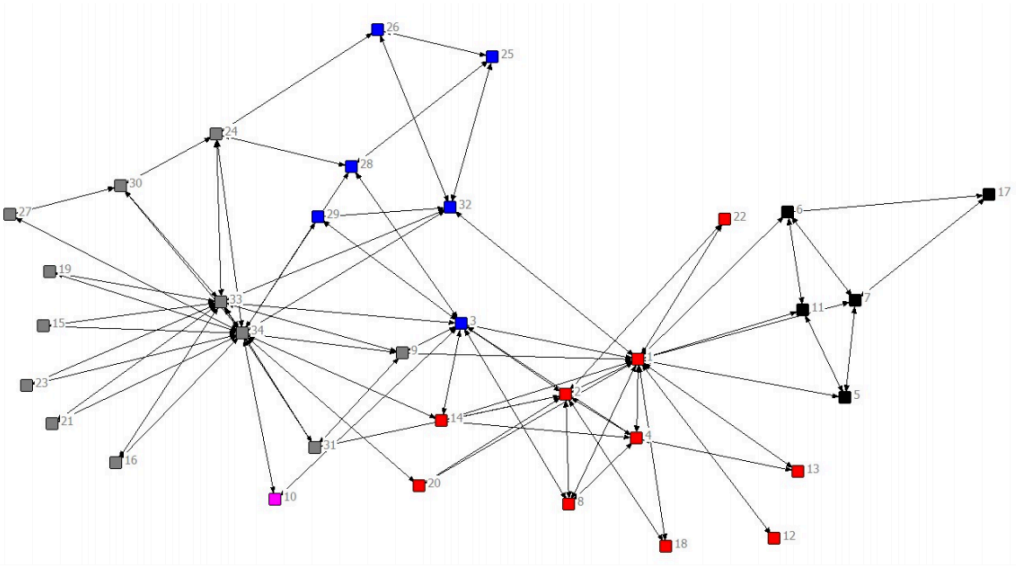

Figure 2.1 - Results of overlapping community detection on the canonical Karate club example. The clusters in red and blue are the overlaps between clusters depicted by colors gray and black.

Figure 2.1 shows results of an overlapping community detection on the Zachary's karate club network [21]. The nodes in blue and red represent the overlaps between clusters of gray and black nodes on the left and right side of the graph respectively.

The link clustering algorithm is one of the standard algorithms in the category of overlapping community detection [6]. In link clustering based algorithms, the edges between the nodes are partitioned and not the nodes themselves. A node is a part of more than one cluster if the edges connecting it are in different clusters [22]. Ahn et al. define a measure called *edge similarity* based upon which they hierarchically cluster the

graph [6]. *Edge Similarity,* given edges e_i and e_j of nodes *i* and *j* incident upon a third node *k,* is defined by the Jaccard coefficient to be:

$$S(e_i, e_j) = \frac{N_i \cap N_j}{N_i \cup N_j}$$

where N_i and N_j are the neighborhoods of nodes *i* and *j*, respectively. This yields a link dendogram depicting the structural hierarchy of the graph. Cutting the links in the dendogram at certain thresholds obtains the community structure [22].

The clique percolation method (CPM) is another established algorithm for overlapping community detection. A *Clique* in graph theory refers to a subset C of vertices V in an undirected graph G = (V, E) such that for every pair of vertices in C, there exists an edge connecting the two [23]. The underlying assumption for CPM is that a community is composed of overlapping sets of fully connected subgraphs and by clustering adjacent communities together.

For a given value of *k,* CPM finds all the cliques of size *k* in a graph. Once all the *k* cliques are identified a new graph is constructed where nodes in one clique are collapsed into one vertex; the connected components of this graph help in detecting which cliques compose a community. The overlap between communities arises because of the fact that a particular node can belong to more than one clique in the graph [22]. For a small value of *k*, i.e. between 3 and 6, Palla et al. have shown decent results [7]. Adamcsek et al. proposed CFinder which is an implementation of CPM and its time complexity for many graphs is polynomial [24]. However, CPM often fails to converge for very large

graphs as it aims to find localized patterns in the network as opposed to global communities [22].

Some other algorithms in this class try to find overlapping communities by using local optimization techniques. Baumes et al. reimagine communities as a subset of actors (nodes) who give rise to a locally optimal subgraph with respect to a density function defined on those actors [25]. This locally optimal subgraph obtained is, in other words, a community and two different subgraphs with significant overlaps can still be optimal at the same time thus giving us the overlapping communities. This algorithm is a two-step process. In the first step, called *RankRemoval,* highly ranked nodes are iteratively removed to form disjoint clusters which are then used as seed communities. In the second step, called *IterativeScan,* the disjoint clusters are expanded by addition or removal of nodes until local optima for the density function is reached [22]. The worst case runtime of this algorithm is O(n^2) while the result obtained is contingent about the *RankRemoval* stage and the clusters obtained to seed the algorithm.

Some other noted algorithms use fuzzy detection techniques. These algorithms try to calculate a membership vector per node where the columns of the vector represent the *belonging factor* of that node for a particular community [22]. These algorithms assume that we know the number of communities in the network beforehand. While this is a drawback, domain knowledge could be used along with experimentation to determine the *k*. One of the prime implementations of fuzzy detection was by Nepusz et al. who

modelled this as a nonlinear constrained optimization problem and used simulated annealing methods to solve it [26].

Gregory et al. build upon Girvan-Newman clustering by using the notion of *split betweenness* to find overlapping communities [27]. Gregory proposes the *Cluster Overlap Newman Girvan Algorithm* (CONGA), which performs hierarchical clustering on the graph like Girvan and Newman's well known algorithm [15], but allows the communities to overlap.

Gregory defines *split betweenness* to be the number of shortest paths passing through a given node if it were to be split [27]. The algorithm removes the edge with maximum edge betweenness or splits the vertex with maximum split betweenness during each iteration. The algorithm converges when no edges are removed or split. The split vertices constitute bridges between communities, thus causing overlaps [27].

In some other interesting approaches, Airoldi et al. build a mixed membership stochastic blockmodel for overlapping community detection [28]. They developed a probabilistic model and describe a fast inference algorithm for inference and estimation (E-M). The model takes into account multiple roles exhibited by nodes in interaction with others. They demonstrated their model quite successfully on protein interaction and social networks, thus affirming its wide applicability. In some other approaches, genetic algorithms were used by Cai et al. for overlapping communities [29].

However, all the algorithms described in this section typically have problems scaling to really large graphs of the type found in the real world. In addition to that, their

computation kernels tend to be complicated and nontrivial to parallelize. Only recently has the research in the area started to acknowledge the problems with scalability. Prat-Pérez et al. combined a multiplicity of methods to obtain a scalable algorithm [9].

Non-negative Matrix Factorization (NMF) is a technique used in machine learning for feature detection and is considered to be a critical component of recommendation systems. The famed *Netflix* problem factorizes a non-negative matrix M into two matrices under some constraint M = FH where F represents the new matrix with the feature set [22]. If the problem is cast in terms of overlapping community detection, each element of F is the dependence of a particular node to a community. Psorakis et al. demonstrate application of NMF in community detection [30]. Using NMF has some benefits; namely, the problem has been well understood in Machine Learning space and a fair amount of work has been done in scaling the implementations of the problem.

Yang and Leskovec, build around NMF and their unique empirical observations about nature of overlaps in communities amongst graphs, which they encapsulate within their algorithm called BigCLAM (Cluster Affiliation Model for BigNetworks) [8]. In their evaluation they demonstrate that BigCLAM outperforms its rivals such as CPM and Link Clustering easily on most benchmarks including scalability to very large graphs. Thus BigCLAM represents the state-of-the-art in overlapping community detection and hence a good candidate for further investigation for scalability.

3. The BigCLAM algorithm

In this section, we describe the BigCLAM algorithm and discuss why it represents the state of the art in overlapping community detection.

3.1 Introduction

Yang and Leskovec present *Community Affiliation Model for BigNetworks* (BigCLAM) based on the counterintuitive empirical observation that the more communities a pair of nodes share, the more likely they are to be connected to each other [8]. Almost all overlapping community detection methods make an assumption that community overlaps are less densely connected than the parts that do not overlap. This implies that the more communities a pair of nodes share, the less likely they are to be connected to each other [8]. However, they find quite the opposite in real world networks; the more the nodes share communities, the more likely they are to be connected.

The authors empirically studied communities with ground truth and realized that on average 85% members of a community belong to another community at the same time; thus, the community intersections are dense as opposed to sparse as proposed by most models [8]. This follows from the fact that for nodes that belong to one or more communities, edges between those nodes often exist for one primary reason. Thus the likelihood of sharing an edge between two nodes increases as the number of communities shared by the nodes increases.

Thus in the words of Yang and Leskovec, BigCLAM encapsulates the following three observations:

1. Shared group affiliations are the cause of emergence of communities.
2. Nodes are involved in a particular community to varying degrees.
3. The more communities a pair of nodes share, the more likely they are to be connected to each other.

With this model in mind; we can define a non-negative matrix *F* such that F_{uc} is a weight between node $u \in V$ and community $c \in C$ for graph *G(V, E)* where an edge exists between nodes *u* and *v* with probability *p(u,v)* and F_u is weight vector for node *u* [8]. Every column of F_u signifies degree of belonging of node *u* to community *c* represented by the column.

3.2 Community Detection using BigCLAM

Given an undirected graph *G(V, E)*, in order to detect *k* communities, Yang and Leskovec fit the affiliation matrix *F* to equation $F \in \mathbb{R}^{N \times K}$ by maximizing likelihood *l(F) = log P(G |F),* thus,

$$F = argmax\ l(F)$$

This equation is a variant of nonnegative matrix factorization (NMF) with log likelihood – *l(F)* as the loss function. While most NMF methods have l_2 norm as the loss function, it does not work as well with adjacency matrices [8]. Also, using the log likelihood as the

loss function means that the gradient can be calculated in near linear time which represents a big speedup over the existing methods [8].

With this information in place to solve the community detection problem, block coordinate descent is used. F_u for each node *u* is fixed vis-à-vis F_v, thus the membership of a node *u* is fixed with respect to all other nodes. Thus F_u becomes a convex optimization problem:

$$\sum_{v \notin N(u)} F_v = \quad (\sum_v F_v - F_u - \sum_{v \in N(u)} F_v) \qquad (3.1)$$

The *N(u)* here represents the set of neighbors of node *u*.

3.2.1 Initialization

F_u is updated during every iteration according to equation 3.1 and the algorithm only converges when likelihood does not increase by more than 0.001%. To estimate *F* by this method however, it needs to be initialized first.

Communities in *F* are initialized by locally minimal conductance as shown by Gleich et al. [31]. They show that locally empirical neighborhoods which minimize conductance are good seed sets for community detection algorithms [8].

3.2.2 Determination of Community Affiliation

After equation 3.1 converges we obtain an affiliation matrix *F.* However we still need to determine whether node *u* belongs to community *c* [8]. Thus we need to decide a threshold *∂* such that for any value below *∂* the community affiliation of node *u* is

ignored if $F_{uc} \leq \partial$. Conversely if $F_{uc} \geq \partial$, then node *u* is determined to be a member of community *c*.

3.3 Algorithm

Thus BigCLAM can be broken down into the following functional steps:

1. For a given *k*, initialize the matrix F where F_u corresponds to the community vector for a node *u* each of whose columns capture the degree of belonging of node *u* to that community represented by the column. Leskovec and Yang seed the original communities using the conductance measure as described in [31].
2. Gradient ascent starts with this step, for each iteration a node is randomly sampled from the graph.
3. Create a temporary vector F_u for the sampled node *u*. This F_u includes only the communities that the neighbors of *u* are a part of. If a there is a community *c* such that *u* is a part of that community, yet none of its neighbors belong to that community, it must be deleted.
4. Calculate the gradient for this new F_u using equation 3.1
5. Calculate the learning rate (step size) by using line search and update F_u with the new values.
6. After every 10,000 or so iterations, likelihood is calculated, and if the difference between the last likelihood value and current likelihood is under 0.001%, the algorithm converges.

While for large graphs, BigCLAM can take a while to converge, it is still many magnitudes of improvement over other community detection algorithms.

3.4 Performance

The authors compared BigCLAM against state-of-the-art implementations of Link Clustering, CPM and Mixed-Membership Stochastic Blockmodel (MMSB). They demonstrated that BigCLAM performed 10 to 100 times faster than their nearest rivals. They also demonstrated the viability of their approach for large scale networks and showed that they could scale BigCLAM to a sample dataset from the LiveJournal network containing 3 million nodes and 34.5 million edges [8].

Thus BigCLAM undoubtedly outperforms all other community detection algorithms owing to the intrinsic scalability of its NMF method, making it a good candidate to test our hypothesis about even further scaling community detection using a vertex centric graph processing framework.

3.5 Qualitative Analysis

As we noted earlier, it is often difficult to present a generalized solution to community detection since we need to apply domain knowledge specific to the particular network being analyzed. However, the latent factor model at the heart of the non-negative matrix factorization in BigCLAM attempts to learn the “hidden factors” that underpin the community affiliations in a given network. These factors are learned during the course of execution of the algorithm, thus the amount of domain specific knowledge

necessary to detect communities with BigCLAM is minimal. These latent or hidden factors map the relationship between different communities and the nodes in the network. Thus, if the network under study conforms to the 3 observations noted on page 14, communities can be detected by BigCLAM regardless of the domain of the problem at hand.

Another area where BigCLAM does well qualitatively is its running time. Running time for modularity calculation at the heart of other popular methods is in the order of $O(nm^2)$ where *n* is the number of nodes while *m* is the number of edges. In contrast, NMF optimization used in BigCLAM is a linear time operation during each iteration, hence BigCLAM can scale to very large networks quite easily [8].

3.6 Possible Improvements

While the fact that explicit domain knowledge isn't required to detect communities is an overall positive, accuracy of communities detected is sensitive to the properties of the nodes of the graph. This is something BigCLAM entirely overlooks. A good way to model node properties and incorporating that into BigCLAM's community detection would be an additional improvement.

One of the other major drawbacks is the fact that one needs to have a good idea of the number of communities, i.e. *k,* in the given dataset. While it is possible to estimate *k* using the best minimal conductance, the process is slow and cumbersome.

Lastly the accuracy of the output and the time it takes for the algorithm to converge depend on the original seed communities used to bootstrap the algorithm. While neighborhoods with locally minimal conductance are good seed sets for the algorithm [8], the seed step needs to be thoroughly evaluated for different problem domains and revised as newer methods are proposed in this area.

4. BigCLAM on GraphChi

In this section we briefly describe the GraphChi framework and its benefits in terms of scaling graph processing, and then we describe a BigCLAM implementation for GraphChi.

4.1 Introduction

Processing and distributing computations on real world graphs, which are in the range of tens of millions of nodes and edges which are at least an order of magnitude greater in size, is a challenging task. Some high level abstractions have been proposed to ease the process of scaling the graph algorithms onto clusters in order to leverage the immediate benefits offered by coarse-grained distribution of tasks over a cluster.

We chose GraphLab for our implementation as it is an open source alternative to Pregel which was the first such graph processing framework developed [11]. GraphLab provides a level of indirection for programmers from the low level abstractions such as MPI or OpenMP. GraphLab uses a graph-based data model to represent both data and computational dependencies. This data model lends itself quite well to represent overlapping community detection algorithms. Additionally, GraphLab takes care of consistency guaranteeing, task scheduling and aggregating the global state of the algorithm [11].

User defined functions in GraphLab are defined as an update function that gets executed on every vertex in the graph for each iteration. Thus GraphLab is both data and graph parallel. The vertices of the graph have access to a globally shared state and they can send and receive messages from their neighbors as well as modify the graph state in the update function.

4.2 BigCLAM Implementation

We re-implemented BigCLAM in GraphChi. GraphChi is a disk-based graph processing cousin of GraphLab's; while Graphlab is meant to run on distributed clusters, GraphChi works on standard multicore computers, making it great for prototyping, and porting GraphChi programs onto GraphLab is trivial because they share their computational model [32].

There are several challenges in porting even the simplest of graph analysis algorithms to a graph processing framework because:

1. It is hard to fit existing algorithms into a vertex centric approach.
2. GraphLab only exposes its graph model with a highly restrictive set of functions such as *update*. But most algorithms have several step and you need to write one function per step.

In case of BigCLAM, it can be broken down into 2 GraphLab programs. The first program is the Conductance computation phase where the matrix F is seeded with community assignments for nodes. The second program is the gradient ascent step.

In gradient ascent step, the *update* function for each vertex contains the main gradient ascent kernel for the algorithm. During an iteration, every node *u* calls the *update* function and re-estimates the vector F_u for that node. Thus instead of nodes being sampled, as in the case of original BigCLAM implementation per iteration, each iteration of GraphChi calls an update on every node in the graph. Although this causes potential unsafe behavior as the shared matrix F is simultaneously updated by all the nodes during each iteration, the idea is that it does not matter eventually as the computation should converge to a steady state although this may cause the algorithm to take more iterations to converge. The log likelihood is calculated after every iteration and the algorithm converges when the difference between two succeeding iterations (the step size) is less than 0.001%.

5. Experimental Evaluation

In this section we compare the original implementation of BigCLAM with our GraphChi version over four real world graphs signifying a broad range of standard workloads.

5.1 Datasets and Hardware

We used a standard 8 core laptop with 16GB RAM for our experiments. The datasets were obtained from *The Stanford Large Network Dataset Collection* [33]. We used 4 graphs containing ground truth from different online social networks. The datasets used were:

1. *com-DBLP*: DBLP is the bibliography of computer science research papers. Two nodes (authors) are connected if they co-author a paper and a ground truth community in this network is defined to be a publishing conference or a journal that the authors published in. The data contains 317080 nodes and 1049866 edges. This is by far the smallest network used in our experiments.

2. *com-YouTube:* This social network was scraped off of YouTube, the video sharing website. It allows users to be friends with each other and also form groups. The user specified groups are the ground truth communities in the experiments. This graph contains about 1134890 nodes and 2987624 edges.

3. *com-Amazon:* This network was created by scraping product listings on Amazon. If two products are frequently co-purchased together there is an undirected edge between the two. The product categories defined by Amazon are the ground truth communities in this network. This is another mid-sized network like YouTube, it has 334863 nodes and 925872 edges.

4. *com-LiveJournal*: This was the largest network used in our experiments and was created by using the data from Live Journal blogging platform. Users can form friendships to each other and join groups, which form the basis of our ground truth communities. This network contains 3997962 nodes (about 4 million) and 34681189 edges (34.6 million).

5.2 Experiments on BigCLAM OpenMP

The standard implementation of BigCLAM is found in *Stanford Network Analysis Project (SNAP) Framework* [33]. It uses optimized non-STL datastructures along with OpenMP to provide parallelism. The OpenMP implementation however in our experience performed best at 16 threads on our 8 core machine with 16GB RAM. We ran the experiments for number of communities, k = 10, 100 and 250. The results are plotted in Figure 5.1.

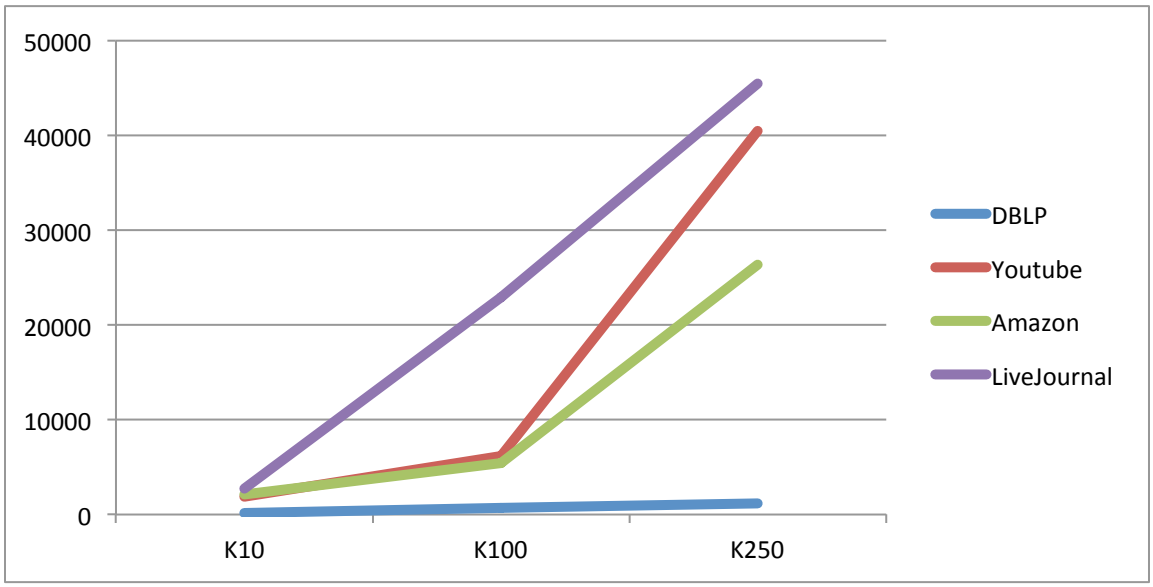

Figure 5.1 - Plot of runtimes (in seconds) for the 4 datasets for different values of k.

BigCLAM performed well for all values of *k* for the *DBLP* dataset. For *Amazon* and *YouTube* it scaled reasonably well up to *k* = 100. For that value, the runtime for *Amazon* dataset was 5442 seconds while the runtime for *YouTube* was 6185 seconds which is roughly under 2 hours. When value of *k* was increased to 250, the runtimes spiked up quite significantly. For *Amazon* it was 26346 seconds while for *YouTube* it was 40481 seconds. *LiveJournal,* which was our biggest dataset containing nodes and edges larger by an order of magnitude, had the longest runtimes naturally. At *k = 100*, it took 22926 seconds which is roughly 4x the runtime for *Amazon* or *YouTube* for that value of *k*. The runtime for *k = 100*, was 45487 seconds.

Thus while BigCLAM scales well to large networks in terms of convergence and detecting meaningful communities, its standard implementation does not scale that well in terms

of running times. Even with 16 OpenMP threads the largest experiment, one for *LiverJournal* with *k* set to 250, took about 12 hours.

5.3 Experiments on BigCLAM GraphChi

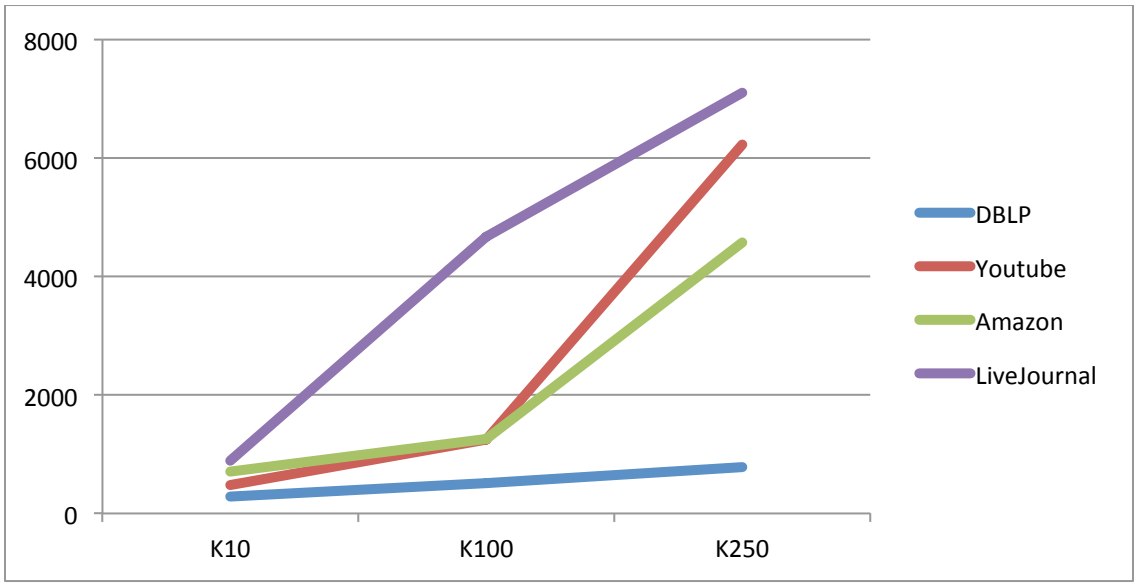

Figure 5. 2 - Plot of runtimes (in seconds) for the 4 datasets for GraphChi version for different values of k.

Except for very trivial examples, GraphChi version significantly outperforms the OpenMP version (see Figure 5.2). The DBLP example was actually faster on OpenMP, while for others we saw significant gains from *k* = 100 onwards. For *Amazon* data set, the running times were 1256 seconds for *k* = 100 and 4573 seconds for *k* = 250 showing 4.33x and 5.76x speedups respectively. For the *YouTube* dataset, the running times were 1244 seconds for *k* = 100 and 6227 seconds for *k* = 250 and the speedups when compared to OpenMP version were 4.97x and 6.5x.

The results were even more stark for the largest dataset, *LiveJournal*. For k = 100, the runtime was 4,669 seconds, down from 22,926 seconds which was 4.91x improvement while for k = 250, the runtime was 7103 seconds down from 45,487 seconds which was speedup of 6.4x.

A non-optimized vertex centric implementation of the original algorithm was able to outperform the parallel OpenMP code with ease. The gains were especially palpable for larger experiments with networks such as *LiveJournal* which took about 12 hours to run. A 6.4x speedup definitely made a huge impact in this case. The performance for DBLP was equivalent or in some cases actually worse than the OpenMP version. Thus GraphChi maybe not be a "one size fits" all solution and it can be best leveraged for very large scale workloads.

Lastly, GraphChi programs can be distributed to multi-node clusters with relative ease. This fact alone makes our approach more compelling as for problems even bigger than the ones we used in our experiments, the solution could in theory, be scaled out on to a cluster. We would like to test in future how well this hypothesis for very large graphs.

6. Conclusion and Future Work

The main contributions of our work were:

1. We provided a vertex based alternative implementation to BigCLAM.
2. Our preliminary experimental results show that leveraging inherent graph and data parallel models of frameworks like GraphChi gave us promising results even when running out on a standard Macbook laptop. GraphChi and its distributed counterpart, GraphLab, can push computational boundaries on very large graph problems.

Furthermore, our approach can be scaled to big compute clusters in the cloud as GraphChi API is entirely compliant with GraphLab. GraphChi's data model provides a powerful abstraction to scale complex optimizations involved in graph processing. Thus transparent linear speedups are possible without rewriting algorithms for a massively distributed framework. Our work could be further extended by implementing and evaluating more recent algorithms such as the one proposed by Prat-Pérez et al. [9].

Lastly, all existing overlapping community detection algorithms could be classified according to the optimization methods used in their computational kernels and these optimization methods could be implemented in GraphChi, thus our work could be expanded into an extensible general purpose toolkit for overlapping community detection at scale.

References

[1] Girvan, M., & Newman, M. E. (2002). “Community structure in social and biological networks.” In *Proceedings of the National Academy of Sciences*, *99*(12), 7821-7826.

[2] Pothen, A. (1997). “Graph partitioning algorithms with applications to scientific computing.” In *Parallel Numerical Algorithms* (pp. 323-368). Springer Netherlands.

[3] Diesner, J., Frantz, T. L., & Carley, K. M. (2005). “Communication networks from the Enron email corpus “It's always about the people. Enron is no different”.” *Computational & Mathematical Organization Theory*, *11*(3), 201-228.

[4] Fortunato, S. (2010) "Community detection in graphs." *Physics Reports* 486.3: 75-174.

[5] Kelley, S., Goldberg, M., Magdon-Ismail, M., Mertsalov, K., and Wallace, A. 2011. “Handbook of Optimization in Complex Networks.” Springer, Chapter 6.

[6] Ahn, Y. Y., Bagrow, J. P., & Lehmann, S. (2010). “Link communities reveal multiscale complexity in networks.” *Nature*, *466*(7307), 761-764.

[7] Palla, G., Derényi, I., Farkas, I., & Vicsek, T. (2005). “Uncovering the overlapping community structure of complex networks in nature and society.” *Nature*, *435*(7043), 814-818.

[8] Yang, J., & Leskovec, J. (2013). “Overlapping community detection at scale: a nonnegative matrix factorization approach.” In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining (pp. 587-596).* ACM.

[9] Prat-Pérez, A., Dominguez-Sal, D., & Larriba-Pey, J. L. (2014, April). “High quality, scalable and parallel community detection for large real graphs.” In *Proceedings of the 23rd International Conference on World Wide Web* (pp. 225-236). International World Wide Web Conferences Steering Committee.

[10] Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., & Czajkowski, G. (2010, June). “Pregel: a system for large-scale graph processing.” In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (pp. 135-146).* ACM.

[11] Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., & Hellerstein, J. M. (2010). “Graphlab: A new framework for parallel machine learning.” *arXiv preprint arXiv:1006.4990*.

[12] http://en.wikipedia.org/wiki/Modularity_(networks)

[13] Newman, M. E. (2006). “Modularity and community structure in networks.” In *Proceedings of the National Academy of Sciences*, 103(23), 8577-8582.

[14] Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z., & Wagner, D. (2006). “On Modularity-NP-Completeness and Beyond.” Univ., Fak. für Informatik, Bibliothek.

[15] Newman, M. E. (2004). “Fast algorithm for detecting community structure in networks.” *Physical Review E*, *69*(6), 066133.

[16] Guimera, R., Sales-Pardo, M., & Amaral, L. A. N. (2004). “Modularity from fluctuations in random graphs and complex networks.” *Physical Review E, 70*(2), 025101.

[17] Duch, J., & Arenas, A. (2005). “Community detection in complex networks using extremal optimization.” *Physical Review E*, *72*(2), 027104.

[18] Newman, M. E. (2006). “Finding community structure in networks using the eigenvectors of matrices.” *Physical Review E*, *74*(3), 036104.

[19] http://en.wikipedia.org/wiki/Random_walk

[20] Zhou, H. (2003). “Distance, Dissimilarity index, and Network community structure.” *Physical Review E*, *67*(6), 061901.

[21] Zachary, W. W. (1977). “An information flow model for conflict and fission in small groups.” *Journal of Anthropological Research,* 452-473.

[22] Xie, J., Kelley, S., & Szymanski, B. K. (2013). “Overlapping community detection in networks: The state-of-the-art and comparative study.” *ACM Computing Surveys* (CSUR), 45(4), 43.

[23] http://en.wikipedia.org/wiki/Clique_(graph_theory)

[24] Adamcsek, B., Palla, G., Farkas, I. J., Derényi, I., & Vicsek, T. (2006). “CFinder: locating cliques and overlapping modules in biological networks.” *Bioinformatics*, *22*(8), 1021-1023.

[25] Baumes, J., Goldberg, M. K., Krishnamoorthy, M. S., Magdon-Ismail, M., & Preston, N. (2005). "Finding communities by clustering a graph into overlapping subgraphs." *IADIS AC*, *5*, 97-104.

[26] Nepusz, T., Petróczi, A., Négyessy, L., & Bazsó, F. (2008). "Fuzzy communities and the concept of bridgeness in complex networks." *Physical Review E*, *77*(1), 016107.

[27] Gregory, S. (2007). "An algorithm to find overlapping community structure in networks." In *Knowledge Discovery in Databases: PKDD 2007* (pp. 91-102). Springer Berlin Heidelberg.

[28] Airoldi, E. M., Blei, D. M., Fienberg, S. E., & Xing, E. P. (2009). "Mixed membership stochastic blockmodels." In *Advances in Neural Information Processing Systems* (pp. 33-40).

[29] Cai, Y., Shi, C., Dong, Y., Ke, Q., & Wu, B. (2011). "A novel genetic algorithm for overlapping community detection." In *Advanced Data Mining and Applications* (pp. 97-108). Springer Berlin Heidelberg.

[30] Psorakis, I., Roberts, S., Ebden, M., & Sheldon, B. (2011). "Overlapping community detection using bayesian non-negative matrix factorization." *Physical Review E*, *83*(6), 066114.

[31] Gleich, D., & Seshadhri, C. (2012). "Neighborhoods are good communities." In *KDD'12*.

[32] Kyrola, A., Blelloch, G. E., & Guestrin, C. (2012, October). "GraphChi: Large-Scale Graph Computation on Just a PC." In *OSDI* (Vol. 12, pp. 31-46).

[33] https://snap.stanford.edu/